转自：http://blog.csdn.net/zeuseign/article/details/72744482

TensorFlow学习笔记（七）：TensorBoard可视化助手

TensorBoard可以将训练过程中的各种绘制数据展示出来，包括标量（scalars），图片（images），音频（Audio）,计算图（graph）,数据分布，直方图（histograms）和嵌入式向量。

使用TensorBoard展示数据，需要在执行Tensorflow就算图的过程中，将各种类型的数据汇总并记录到日志文件中。然后使用TensorBoard读取这些日志文件，解析数据并生产数据可视化的Web页面，让我们可以在浏览器中观察各种汇总数据。

summary_op包括了summary.scalar、summary.histogram、summary.image等操作，这些操作输出的是各种summary protobuf，最后通过summary.writer写入到event文件中。

Tensorflow API中包含系列生成summary数据的API接口，这些函数将汇总信息存放在protobuf中，以字符串形式表达。

对标量数据汇总和记录使用tf.summary.scalar，函数格式如下：

```
[python]
tf.summary.scalar(tags, values, collections=None, name=None)
```

使用tf.summary.histogram直接记录变量var的直方图，输出带直方图的汇总的protobuf，函数格式如下：

```
[python]
tf.summary.histogram(tag, values, collections=None, name=None)
```

输出带图像的probuf，汇总数据的图像的的形式如下：' *tag* /image/0', ' *tag* /image/1', etc.，如：input/image/0等

```
[python]
tf.summary.image(tag, tensor, max_images=3, collections=None, name=None)
```

将上面几种类型的汇总再进行一次合并，具体合并哪些由inputs指定，格式如下：

```
tf.summary.merge(inputs, collections=None, name=None)
```

合并默认图形中的所有汇总：

```
[python]
tf.summaries.merge_all(key='summaries')
```

将汇总的protobuf写入到event文件中去的相关的类： SummaryWriter是一个类，它可以调用以下成员函数

来往event文件中添加相关的数据 add*summary(), add* session*log(), add*_event(), or add_graph()

```
[python]
tf.summary.FileWriter
```

这里注意，计算图形的信息通过add_graph写入到event文件中。

下面通过MNIST代码例子讲解各种类型数据的汇总和展示的方法。

```
[python]
# coding=utf-8

import tensorflow as tf
"""
首先载入Tensorflow，并设置训练的最大步数为1000,学习率为0.001,dropout的保留比率为0.9。
同时，设置MNIST数据下载地址data_dir和汇总数据的日志存放路径log_dir。
这里的日志路径log_dir非常重要，会存放所有汇总数据供Tensorflow展示。
"""

from tensorflow.examples.tutorials.mnist import input_data
max_step = 1000
learning_rate = 0.001
dropout = 0.9
data_dir = '/tmp/tensorflow/mnist/input_data'
log_dir = 'tmp/tensorflow/mnist/logs/mnist_with_summaries'

# 使用input_data.read_data_sets下载MNIST数据，并创建Tensorflow的默认Session
mnist = input_data.read_data_sets(data_dir, one_hot=True)
sess = tf.InteractiveSession()

"""
为了在TensorBoard中展示节点名称，设计网络时会常使用tf.name_scope限制命名空间，
在这个with下所有的节点都会自动命名为input/xxx这样的格式。
定义输入x和y的placeholder，并将输入的一维数据变形为28×28的图片存储到另一个tensor，
这样就可以使用tf.summary.image将图片数据汇总给TensorBoard展示了。
"""
with tf.name_scope('input'):
    x = tf.placeholder(tf.float32, [None, 784], name='x_input')
    y = tf.placeholder(tf.float32, [None, 10], name='y_input')

with tf.name_scope('input_reshape'):
    image_shaped_input = tf.reshape(x, [-1, 28, 28, 1])
    tf.summary.image('input', image_shaped_input, 10)

# 定义神经网络模型参数的初始化方法，
# 权重依然使用常用的truncated_normal进行初始化，偏置则赋值为0.1
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# 定义对Variable变量的数据汇总函数
"""
计算出Variable的mean,stddev,max和min，
对这些标量数据使用tf.summary.scalar进行记录和汇总。
同时，使用tf.summary.histogram直接记录变量var的直方图。
"""
def variable_summaries(var):
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var-mean)))
        tf.summary.scalar('stddev', stddev)
        tf.summary.scalar('max', tf.reduce_max(var))
        tf.summary.scalar('min', tf.reduce_min(var))
        tf.summary.histogram('histogram', var)

# 设计一个MLP多层神经网络来训练数据，在每一层中都会对模型参数进行数据汇总。
"""
定一个创建一层神经网络并进行数据汇总的函数nn_layer。
```

```
这个函数的输入参数有输入数据input_tensor,输入的维度input_dim,输出的维度output_dim和层名称layer_name
在函数内，显示初始化这层神经网络的权重和偏置，并使用前面定义的variable_summaries对variable进行数据汇总
然后对输入做矩阵乘法并加上偏置，再将未进行激活的结果使用tf.summary.histogram统计直方图。
同时，在使用激活函数后，再使用tf.summary.histogram统计一次。
"""
def nn_layer(input_tensor, input_dim, output_dim, layer_name,act=tf.nn.relu):
    with tf.name_scope(layer_name):
        with tf.name_scope('weight'):
            weights = weight_variable([input_dim, output_dim])
            variable_summaries(weights)
        with tf.name_scope('biases'):
            biases = bias_variable([output_dim])
            variable_summaries(biases)
        with tf.name_scope('Wx_plus_b'):
            preactivate = tf.matmul(input_tensor, weights) + biases
            tf.summary.histogram('pre_activations', preactivate)
        activations = act(preactivate, name='actvations')
        tf.summary.histogram('activations', activations)
        return activations

"""
使用刚定义好的nn_layer创建一层神经网络，输入维度是图片的尺寸（784=24×24），输出的维度是隐藏节点数500.
再创建一个Droput层，并使用tf.summary.scalar记录keep_prob。然后再使用nn_layer定义神经网络的输出层，激
在后面会处理。
"""
hidden1 = nn_layer(x, 784, 500, 'layer1')

with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    tf.summary.scalar('dropout_keep_probability', keep_prob)
    dropped = tf.nn.dropout(hidden1, keep_prob)

y1 = nn_layer(dropped, 500, 10, 'layer2', act=tf.identity)

"""
这里使用tf.nn.softmax_cross_entropy_with_logits()对前面输出层的结果进行softmax处理并计算交叉熵损失c
计算平均损失，并使用tf.summary.saclar进行统计汇总。
"""
with tf.name_scope('cross_entropy'):
    diff = tf.nn.softmax_cross_entropy_with_logits(logits=y1, labels=y)
    with tf.name_scope('total'):
        cross_entropy = tf.reduce_mean(diff)
tf.summary.scalar('cross_entropy', cross_entropy)

"""
使用Adma优化器对损失进行优化，同时统计预测正确的样本数并计算正确率accuray，
再使用tf.summary.scalar对accuracy进行统计汇总。
"""
with tf.name_scope('train'):
    train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)
with tf.name_scope('accuracy'):
    with tf.name_scope('correct_prediction'):
        correct_prediction = tf.equal(tf.argmax(y1, 1), tf.arg_max(y, 1))
    with tf.name_scope('accuracy'):
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', accuracy)

"""
由于之前定义了非常多的tf.summary的汇总操作，一一执行这些操作态麻烦，
所以这里使用tf.summary.merger_all()直接获取所有汇总操作，以便后面执行。
然后，定义两个tf.summary.FileWrite(文件记录器)在不同的子目录，分别用来存放训练和测试的日志数据。
同时，将Session的计算图sess.graph加入训练过程的记录器，这样在TensorBoard的GRAPHS窗口中就能展示整个计
最后使用tf.global_variables_initializer().run()初始化全部变量。
"""
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(log_dir + '/train', sess.graph)
test_writer = tf.summary.FileWriter(log_dir + '/test')
tf.global_variables_initializer().run()

"""
定义feed_dict的损失函数。
该函数先判断训练标记，如果训练标记为true,则从mnist.train中获取一个batch的样本，并设置dropout值;
如果训练标记为False，则获取测试数据，并设置keep_prob为1,即等于没有dropout效果。
"""
def feed_dict(train):
    if train:
        xs, ys = mnist.train.next_batch(100)
        k = dropout
    else:
```

```
        xs, ys = mnist.test.images, mnist.test.labels
        k = 1.0
    return {x: xs, y: ys, keep_prob: k}

# 实际执行具体的训练，测试及日志记录的操作
"""
首先，使用tf.train.Saver()创建模型的保存器。
然后，进入训练的循环中，每隔10步执行一次merged（数据汇总），accuracy（求测试集上的预测准确率）操作，
并使应test_write.add_summary将汇总结果summary和循环步数i写入日志文件；
同时每隔100步，使用tf.RunOption定义Tensorflow运行选项，其中设置trace_level为FULL—TRACE，
并使用tf.RunMetadata()定义Tensorflow运行的元信息，
这样可以记录训练是运算时间和内存占用等方面的信息.
再执行merged数据汇总操作和train_step训练操作，将汇总summary和训练元信息run_metadata添加到train_writ
平时，则执行merged操作和train_step操作，并添加summary到trian_writer。
所有训练全部结束后，关闭train_writer和test_writer。
"""
saver = tf.train.Saver()
for i in range(max_step):
    if i % 10 == 0:
        summary, acc = sess.run([merged, accuracy], feed_dict=feed_dict(False))
        test_writer.add_summary(summary, i)
        print('Accuracy at step %s: %s' % (i, acc))
    else:
        if i % 100 == 99:
            run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRA(
            run_metadata = tf.RunMetadata()
            summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(True),
                                  options=run_options, run_metadata=run_metadata)
            train_writer.add_run_metadata(run_metadata, 'step%03d' % i)
            train_writer.add_summary(summary, i)
            saver.save(sess, log_dir+"/model.ckpt", i)
            print('Adding run metadata for', i)
        else:
            summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(True))
            train_writer.add_summary(summary, i)
train_writer.close()
test_writer.close()
```

之后切换到Linux终端命令下，执行TensorBoard程序，并通过--logdir指定TensorFlow日志路径，然后哦=TensorBoard就可以自动生成所有汇总数据可视化的结果来。

```
[python]
tensorboard --logdir=/tmp/tensorflow/mnist/logs/mnist_with_summaries
```

执行上面的命令后，出现一条提示信息，复制其中的网址到浏览器，就可以看到数据可视化的图标来。

```
[python]
Starting TensorBoard b'39' on port 6006
(You can naviiage to http://0.0.0.0.6006)
```

以下是tensorFlow1.0以下版本的方式，可供其他版本的同学借鉴下：

使用tf.scalar_summary来收集想要显示的变量，使用scalar_summary的时候，注意tag和tensor的shape一致，tf.scalar_summary(节点名称，获取的数据)，例如:下文代码实例中的loss以及accurary都可以使用。

各层网络权重、偏置的分布，用histogram_summary函数。

historgram_summary用于生成分布图，也可以用saclar_summary记录存数值；前者在history一栏里查看分布图，后者在event一栏中查看数值变化情况。

当需要获取的数据较多的时候，我们一个一个去保存获取到的数据，以及一个一个去运行会显得比较麻烦。

tensorflow提供了一个简单的方法，就是合并所有的summary data的获取函数，保存和运行只对一个对象

进行操作。比如，写入默认路径中，比如/tmp/mnist_logs (by default)

定义一个summury op, 用来汇总多个变量

```python
merged = tf.merge_all_summaries()
```

得到一个summy writer，指定写入路径

```python
tf.train.SummaryWriter()
```

添加写入

```python
train_writer.add_summary()
```

以下为完整实例代码：

```python
"""
Please note, this code is only for python 3+. If you are using python 2+, please modify the co
"""
import tensorflow as tf
import numpy as np


def add_layer(inputs, in_size, out_size, n_layer, activation_function=None):
    # add one more layer and return the output of this layer
    layer_name = 'layer%s' % n_layer
    with tf.name_scope(layer_name):
        with tf.name_scope('weights'):
            Weights = tf.Variable(tf.random_normal([in_size, out_size]), name='W')
            tf.histogram_summary(layer_name + '/weights', Weights)
        with tf.name_scope('biases'):
            biases = tf.Variable(tf.zeros([1, out_size]) + 0.1, name='b')
            tf.histogram_summary(layer_name + '/biases', biases)
        with tf.name_scope('Wx_plus_b'):
            Wx_plus_b = tf.add(tf.matmul(inputs, Weights), biases)
        if activation_function is None:
            outputs = Wx_plus_b
        else:
            outputs = activation_function(Wx_plus_b, )
        tf.histogram_summary(layer_name + '/outputs', outputs)
        return outputs


# Make up some real data
x_data = np.linspace(-1, 1, 300)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape)
y_data = np.square(x_data) - 0.5 + noise

# define placeholder for inputs to network
with tf.name_scope('inputs'):
    xs = tf.placeholder(tf.float32, [None, 1], name='x_input')
    ys = tf.placeholder(tf.float32, [None, 1], name='y_input')

# add hidden layer
l1 = add_layer(xs, 1, 10, n_layer=1, activation_function=tf.nn.relu)
# add output layer
prediction = add_layer(l1, 10, 1, n_layer=2, activation_function=None)

# the error between prediciton and real data
with tf.name_scope('loss'):
    loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),
                                        reduction_indices=[1]))
    tf.scalar_summary('loss', loss)

with tf.name_scope('train'):
    train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```

```
51.
52. sess = tf.Session()
53. merged = tf.merge_all_summaries()
54. writer = tf.train.SummaryWriter("logs/", sess.graph)
55. # important step
56. sess.run(tf.initialize_all_variables())
57.
58. for i in range(1000):
59.     sess.run(train_step, feed_dict={xs: x_data, ys: y_data})
60.     if i % 50 == 0:
61.         result = sess.run(merged, feed_dict={xs: x_data, ys: y_data})
62.         writer.add_summary(result, i)
```

接下来，程序开始运行以后，跑到shell里运行，打开终端，输入如下语句：

cd到指定的文件下

tensorboard --logdir = ‘logs/’

开始运行tensorboard。接下来打开浏览器，进入127.0.0.1：6006 就能够看到loss值在训练中的变化了。